Assignment 2 - Cache Optimization

Computer Systems Organization

Mihir Bani, 2019113003

Note: I completed this assignment with IIIT server Abacus, and not on my local computer. So for Task 0, I have given information from my own laptop, but the stats and relevant information provided in Task 1 and 2 is from running the codes/tools on the server node as well as local pc.

Task 0: System Details

Note: I use WSL (Windows Subsystem for Linux) and some information is not available in the following commands, that are meant to be for Linux. Information like ‘storage’, ‘kernel module’ etc are not available.

- CPU information by running command `` `lscpu` ``

```
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           158
Model name:                      Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
Stepping:                        10
CPU MHz:                         2304.005
BogoMIPS:                        4608.01
Hypervisor vendor:               Microsoft
Virtualization type:             full
L1d cache:                       128 KiB
L1i cache:                       128 KiB
L2 cache:                        1 MiB
L3 cache:                        8 MiB
Vulnerability Itlb multihit:     KVM: Vulnerable
Vulnerability L1tf:              Mitigation; PTE Inversion
Vulnerability Mds:               Vulnerable: Clear CPU buffers attempted, no microcode; SMT Host state unknown
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Full generic retpoline, IBPB conditional, IBRS_FW, STIBP conditional, RSB filling
Vulnerability Srbds:             Unknown: Dependent on hypervisor status
Vulnerability Tsx async abort:   Not affected
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb
rdtscp lm constant_tsc rep_good nopl xtopology cpuid pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_
lm abm 3dnowprefetch invpcid_single pti ssbd ibrs ibpb stibp fsgsbase bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap clflushopt xsaveopt xsavec xgetbv1 xs
aves flush_l1d arch_capabilities
```

- System information obtained by `hardinfo` for WSL system .
    - **Operating system**
      And **Kernel Module = 5.4.72-microsoft-standard-WSL2**

```
Operating System
----------------

-Version-
Kernel          : Linux 5.4.72-microsoft-standard-WSL2 (x86_64)
Version         : #1 SMP Wed Oct 28 23:40:43 UTC 2020
C Library               : GNU C Library / (Ubuntu GLIBC 2.31-0ubuntu9.2) 2.31
Distribution            : Ubuntu 20.04.2 LTS
-Current Session-
Computer Name           : Mihir-Asus
User Name               : mihir ()
Language                : C.UTF-8 (C.UTF-8)
Home Directory          : /home/mihir
-Misc-
Uptime          : 2 days 9 hours 9 minutes
Load Average            : 0.08, 0.10, 0.10
Available entropy in /dev/random                : 3601 bits (healthy)
```

    - **File systems**

```
Filesystems
-----------

-Mounted File Systems-
/dev/sdb        /       8.34 % (230.1 GiB of 251.0 GiB)
tmpfs   /mnt/wsl        0.00 % (3.0 GiB of 3.0 GiB)
tools   /init   67.06 % (78.2 GiB of 237.4 GiB)
none    /dev    0.00 % (3.0 GiB of 3.0 GiB)
none    /run    0.00 % (3.0 GiB of 3.0 GiB)
none    /run/lock       0.00 % (3.0 GiB of 3.0 GiB)
none    /run/shm        0.00 % (3.0 GiB of 3.0 GiB)
none    /run/user       0.00 % (3.0 GiB of 3.0 GiB)
tmpfs   /sys/fs/cgroup  0.00 % (3.0 GiB of 3.0 GiB)
C:\134  /mnt/c  67.06 % (78.2 GiB of 237.4 GiB)
D:\134  /mnt/d  55.67 % (105.3 GiB of 237.6 GiB)
E:\134  /mnt/e  85.94 % (43.5 GiB of 309.5 GiB)
F:\134  /mnt/f  78.81 % (65.6 GiB of 309.5 GiB)
```

    - **Processor:**

```
-Processors-
Package Information
Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz        0       0:0     2304.00 MHz
Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz        1       0:0     2304.00 MHz
Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz        2       0:1     2304.00 MHz
Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz        3       0:1     2304.00 MHz
Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz        4       0:2     2304.00 MHz
Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz        5       0:2     2304.00 MHz
Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz        6       0:3     2304.00 MHz
Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz        7       0:3     2304.00 MHz
```

- **Memory:**

```
Memory
------

-Memory-
MemTotal        Total Memory    6376400 KiB
MemFree Free Memory     3640072 KiB
MemAvailable            5824480 KiB
Buffers         406056 KiB
Cached          1710456 KiB
SwapCached      Cached Swap     212 KiB
Active          1289640 KiB
Inactive                901604 KiB
Active(anon)            49088 KiB
Inactive(anon)          25700 KiB
Active(file)            1240552 KiB
Inactive(file)          875904 KiB
Unevictable             0 KiB
Mlocked         0 KiB
SwapTotal       Virtual Memory  2097152 KiB
SwapFree        Free Virtual Memory     2096628 KiB
Dirty           4 KiB
Writeback               0 KiB
AnonPages               74908 KiB
Mapped          22372 KiB
Shmem           64 KiB
KReclaimable            423740 KiB
Slab            464808 KiB
SReclaimable            423740 KiB
SUnreclaim              41068 KiB
KernelStack             4468 KiB
PageTables              1832 KiB
NFS_Unstable            0 KiB
```

```
Bounce          0 KiB
WritebackTmp            0 KiB
CommitLimit             5285352 KiB
Committed_AS            166556 KiB
VmallocTotal            -1 KiB
VmallocUsed             24088 KiB
VmallocChunk            0 KiB
Percpu          2336 KiB
AnonHugePages           0 KiB
ShmemHugePages          0 KiB
ShmemPmdMapped          0 KiB
FileHugePages           0 KiB
FilePmdMapped           0 KiB
HugePages_Total         0
HugePages_Free          0
HugePages_Rsvd          0
HugePages_Surp          0
Hugepagesize            2048 KiB
Hugetlb         0 KiB
DirectMap4k             107520 KiB
DirectMap2M             4382720 KiB
DirectMap1G             3145728 KiB
```

- **PCI Devices*, USB Devices*, Sensors* and Storage*. DMI* (Desktop Management Interface)**
  (* the list is empty as WSL does not support such devices, or is not supported by `hardinfo`, WSL doesnt have GUI, and thus no DMI also.)

```
PCI Devices
-----------

-PCI Devices-

USB Devices
-----------

-USB Devices-
```

```
Sensors
-------

-Sensors-

Input Devices
-------------

-Input Devices-

Storage
-------
```

```
DMI
---

-No DMI information-
There was an error retrieving the information.
Please try running HardInfo as root.
```

- So attaching some information from Windows 10 OS.
    - **Storage**

Item	Value
Description	Disk drive
Manufacturer	(Standard disk drives)
Model	KINGSTON RBUSNS8154P3256GJ
Bytes/Sector	512
Media Loaded	Yes
Media Type	Fixed hard disk
Partitions	3
SCSI Bus	0
SCSI Logical Unit	0
SCSI Port	1
SCSI Target ID	0
Sectors/Track	63
Size	238.47 GB (256,052,966,400 bytes)
Total Cylinders	31,130
Total Sectors	500,103,450
Total Tracks	7,938,150
Tracks/Cylinder	255
Partition	Disk #1, Partition #0
Partition Size	260.00 MB (272,629,760 bytes)
Partition Starting Offset	1,048,576 bytes
Partition	Disk #1, Partition #1
Partition Size	237.42 GB (254,930,132,480 bytes)
Partition Starting Offset	290,455,552 bytes
Partition	Disk #1, Partition #2
Partition Size	800.00 MB (838,860,800 bytes)
Partition Starting Offset	255,221,301,248 bytes

Description	Disk drive
Manufacturer	(Standard disk drives)
Model	ST1000LX015-1U7172
Bytes/Sector	512
Media Loaded	Yes
Media Type	Fixed hard disk
Partitions	4
SCSI Bus	4
SCSI Logical Unit	0
SCSI Port	0
SCSI Target ID	0
Sectors/Track	63
Size	931.51 GB (1,000,202,273,280 bytes)
Total Cylinders	121,601
Total Sectors	1,953,520,065
Total Tracks	31,008,255
Tracks/Cylinder	255
Partition	Disk #0, Partition #0
Partition Size	237.59 GB (255,112,715,776 bytes)
Partition Starting Offset	1,048,576 bytes
Partition	Disk #0, Partition #1
Partition Size	309.49 GB (332,308,414,464 bytes)
Partition Starting Offset	335,585,214,464 bytes
Partition	Disk #0, Partition #2
Partition Size	309.49 GB (332,308,414,464 bytes)
Partition Starting Offset	667,894,677,504 bytes
Partition	Disk #0, Partition #3
Partition Size	74.94 GB (80,470,867,968 bytes)
Partition Starting Offset	255,114,346,496 bytes

    - **USB devices**

Device	PNP Device ID
Intel(R) USB 3.1 eXtensible Host Controller - 1.10 (Microsoft)	PCI\VEN_8086&DEV_A36D&SUBSYS...

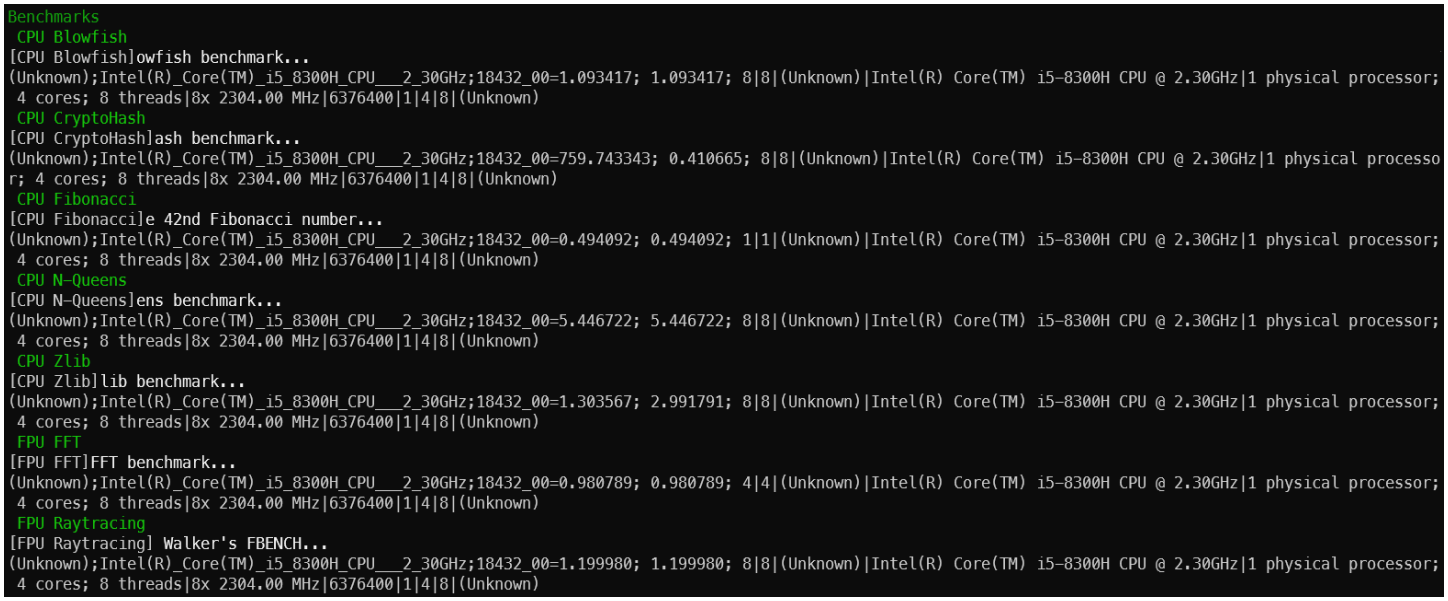
- **Battery:**

```
Battery
-------

-Battery: BAT1-
State           : Unknown
Capacity                : 80 / Normal
Battery Technology              : Unknown
Manufacturer
Model Number            : Microsoft Hyper-V Virtual BatterVirtual
Serial Number           : Virtual
```

- **Benchmarks :**

CPU Blowfish	1.097815
CPU CryptoHash	1104.581180
CPU Fibonacci	0.512099
CPU N-Queens	5.281027
CPU Zlib	1.313949
FPU FFT	0.994809
FPU Raytracing	1.230104



Task 1: Matrix Multiplication

Taking input from a random number generator file and storing output into a file.

- Original code run time
  Running for size = 3500 x 3500 for both the matrices.
  Execution time for matmul = 310.477654
  Total Execution time = 312.193961

- Running for size = 1200 x 1200 for both the matrices.
  Execution time for matmul = 5.914727
  Total Execution time = 6.106516

- Valgrind report :

```
==27663==
==27663== I   refs:      39,308,378,005
==27663== I1  misses:             1,335
==27663== LLi misses:             1,324
==27663== I1  miss rate:           0.00%
==27663== LLi miss rate:           0.00%
==27663==
==27663== D   refs:      21,218,798,080  (21,038,655,186 rd   + 180,142,894 wr)
==27663== D1  misses:     1,838,796,132  ( 1,837,173,581 rd   +   1,622,551 wr)
==27663== LLd misses:           457,594  (       184,193 rd   +     273,401 wr)
==27663== D1  miss rate:            8.7% (           8.7%     +         0.9%  )
==27663== LLd miss rate:            0.0% (           0.0%     +         0.2%  )
==27663==
==27663== LL refs:        1,838,797,467  ( 1,837,174,916 rd   +   1,622,551 wr)
==27663== LL misses:            458,918  (       185,517 rd   +     273,401 wr)
==27663== LL miss rate:             0.0% (           0.0%     +         0.2%  )
```

- Perf Report :

```
+   99.49%     0.00%  original  libc-2.17.so      [.] __libc_start_main
+   99.49%     0.00%  original  original          [.] main
+   98.11%    97.92%  original  original          [.] matmul
+    0.95%     0.03%  original  libc-2.17.so      [.] fprintf
     0.48%     0.48%  original  libc-2.17.so      [.] vfprintf
     0.39%     0.39%  original  [kernel]          [k] 0xffffffffb8f75ed0
     0.39%     0.00%  original  libc-2.17.so      [.] __GI___libc_write
     0.29%     0.29%  original  libc-2.17.so      [.] _itoa_word
     0.22%     0.22%  original  [kernel]          [k] 0xffffffffb8f6c4d6
```

NOTE: the time mentioned with the following cases is taken by averaging over running the same code 3-4 times. As the first time a code is executed it takes more time due to Cold cache miss. But in subsequent runs, the cold cache miss is improved and runtime decreases, so I am only considering from after the 2nd code run.

Optimizations :

For doing optimization, I took both the matrices of size 1200 x 1200. The time measurements are given by running the files on my local pc and not on the Server. The other stats from Perf and Valgrind is recorded by running it on the Server on the same codes.

1. **Ordering of the loop to *ikj* :**

   By this there is a greater chance of cache hit and lower chance of miss. As we are making use of Spatial Locality. This code is much more optimized than the original in which we were not exploiting the spatial locality of the matrices, it means that inside the loop the location in the memory is changing with stride-1 and so it's much faster than other cases with stride>1 .

   Execution time for matmul = 5.057156

   Total Execution time = 5.258106

   CODE:

```
for (i = 0; i < N; ++i)
    for (k = 0; k < N; ++k)
        for (j = 0; j < N; ++j)
            res[i][j] += mat1[i][k] * mat2[k][j];
```

   Perf report showing comparison with other permutations of the loops:

   It shows that the least amount is taken by the *ikj* loop order. The original code on the other hand was of the type *ijk*, and it can be clearly seen here that it was much worse than the optimized version.

```
+   23.47%    23.40%  my_matmul  my_matmul          [.] mult_jki
+   22.75%    22.67%  my_matmul  my_matmul          [.] mult_kji
+   20.59%    20.52%  my_matmul  my_matmul          [.] mult_ijk
+    9.39%     9.34%  my_matmul  my_matmul          [.] mult_jik
+    9.16%     9.12%  my_matmul  my_matmul          [.] mult_kij
+    8.63%     8.52%  my_matmul  my_matmul          [.] mult_ikj
```

   VALGRIND:

```
==27671== I   refs:      66,941,973,354
==27671== I1  misses:             1,329
==27671== LLi misses:             1,318
==27671== I1  miss rate:           0.00%
==27671== LLi miss rate:           0.00%
==27671==
==27671== D   refs:      33,306,156,396  (31,400,894,008 rd   + 1,905,262,388 wr)
==27671== D1  misses:       109,537,927  (   109,355,372 rd   +       182,555 wr)
==27671== LLd misses:           457,616  (       275,111 rd   +       182,505 wr)
==27671== D1  miss rate:            0.3% (           0.3%     +           0.0%  )
==27671== LLd miss rate:            0.0% (           0.0%     +           0.0%  )
==27671==
==27671== LL refs:          109,539,256  (   109,356,701 rd   +       182,555 wr)
==27671== LL misses:            458,934  (       276,429 rd   +       182,505 wr)
==27671== LL miss rate:             0.0% (           0.0%     +           0.0%  )
```

   PERF:

```
+   99.75%     0.00%  ikj      libc-2.17.so      [.] __libc_start_main
+   99.75%     0.00%  ikj      ikj               [.] main
+   97.48%    97.40%  ikj      ikj               [.] matmul
+    1.68%     0.04%  ikj      libc-2.17.so      [.] fprintf
+    0.88%     0.88%  ikj      libc-2.17.so      [.] vfprintf
+    0.51%     0.51%  ikj      libc-2.17.so      [.] _itoa_word
     0.21%     0.21%  ikj      libc-2.17.so      [.] _IO_file_xsputn@@GLIBC_2.2.5
     0.20%     0.18%  ikj      ikj               [.] takeinput
     0.15%     0.15%  ikj      ikj               [.] print_matrix
```

2. **Converting all post-increment/decrement to pre-increment/decrement operator.**

```
++i; // Preferred
i++; // Slower
```

This is faster because in case of post op, the value is first copied and then the operation takes place, whereas in case of pre-op, there is no overhead of copying the variable value as the variable is instantly updated without the need of previous value.

Execution time for matmul = 4.790539

Total Execution time = 4.973955

VALGRIND:

```
==27696== I   refs:      66,941,977,976
==27696== I1  misses:             1,336
==27696== LLi misses:             1,325
==27696== I1  miss rate:           0.00%
==27696== LLi miss rate:           0.00%
==27696==
==27696== D   refs:      33,306,158,064  (31,400,895,178 rd   + 1,905,262,886 wr)
==27696== D1  misses:       109,537,927  (   109,355,373 rd   +       182,554 wr)
==27696== LLd misses:           457,615  (       275,111 rd   +       182,504 wr)
==27696== D1  miss rate:            0.3% (           0.3%     +           0.0%  )
==27696== LLd miss rate:            0.0% (           0.0%     +           0.0%  )
==27696==
==27696== LL refs:          109,539,263  (   109,356,709 rd   +       182,554 wr)
==27696== LL misses:            458,940  (       276,436 rd   +       182,504 wr)
==27696== LL miss rate:             0.0% (           0.0%     +           0.0%  )
```

PERF:

```
+   99.38%     0.00%  pre      libc-2.17.so      [.] __libc_start_main
+   99.38%     0.00%  pre      pre               [.] main
+   97.62%    97.30%  pre      pre               [.] matmul
+    1.21%     0.04%  pre      libc-2.17.so      [.] fprintf
+    0.65%     0.65%  pre      libc-2.17.so      [.] vfprintf
     0.44%     0.44%  pre      [kernel]          [k] 0xffffffffb8f75ed0
     0.44%     0.00%  pre      libc-2.17.so      [.] __GI___libc_write
     0.36%     0.36%  pre      libc-2.17.so      [.] _itoa_word
     0.36%     0.36%  pre      [kernel]          [k] 0xffffffffb8f6c4d6
     0.24%     0.21%  pre      pre               [.] takeinput
     0.15%     0.15%  pre      libc-2.17.so      [.] _IO_file_xsputn@@GLIBC_2.2.5
     0.10%     0.10%  pre      pre               [.] print_matrix
```

3. **Register keyword with the variables.** This makes the variable to be stored in the register directly. This is helpful when the variable is used very frequently, for example counter in a loop.

Execution time for matmul = 3.593632

Total Execution time = 3.787293

Change:

```
register int i = 0;     // Preferred
int i = 0;              // previous
```

```
// Example
register int i = 0;
register int j = 0;
register int k = 0;
```

VALGRIND:

```
==27713==
==27713== I   refs:      51,374,127,260
==27713== I1  misses:             1,335
==27713== LLi misses:             1,324
==27713== I1  miss rate:           0.00%
==27713== LLi miss rate:           0.00%
==27713==
==27713== D   refs:      16,003,097,702  (14,099,280,799 rd   + 1,903,816,903 wr)
==27713== D1  misses:       109,537,932  (   109,355,380 rd   +       182,552 wr)
==27713== LLd misses:           457,614  (       275,113 rd   +       182,501 wr)
==27713== D1  miss rate:            0.7% (           0.8%     +           0.0%  )
==27713== LLd miss rate:            0.0% (           0.0%     +           0.0%  )
==27713==
==27713== LL refs:          109,539,267  (   109,356,715 rd   +       182,552 wr)
==27713== LL misses:            458,938  (       276,437 rd   +       182,501 wr)
==27713== LL miss rate:             0.0% (           0.0%     +           0.0%  )
```

PERF:

```
+   99.25%     0.00%  register  libc-2.17.so      [.] __libc_start_main
+   99.25%     0.00%  register  register          [.] main
+   96.70%    96.44%  register  register          [.] matmul
+    1.85%     0.06%  register  libc-2.17.so      [.] fprintf
+    1.01%     1.01%  register  libc-2.17.so      [.] vfprintf
     0.53%     0.53%  register  [kernel]          [k] 0xffffffffb8f75ed0
+    0.52%     0.00%  register  libc-2.17.so      [.] __GI___libc_write
+    0.52%     0.51%  register  libc-2.17.so      [.] _itoa_word
     0.32%     0.32%  register  [kernel]          [k] 0xffffffffb8f6c4d6
     0.25%     0.21%  register  register          [.] takeinput
     0.20%     0.20%  register  libc-2.17.so      [.] _IO_file_xsputn@@GLIBC_2.2.5
     0.14%     0.14%  register  register          [.] print_matrix
```

4. **Caching inside the matrix multiplication loop.** The value of the matrix which is not changing in the innermost loop, is kept in a temporary variable in the second inner loop. This saves time as the memory is not accessed very frequently as the value is stored in a variable already.
   Execution time for matmul = 3.029244
   Total Execution time = 3.223919

   CODE:

```
for (i = 0; i < m; ++i){
    for (k = 0; k < n; ++k){
      temp_i_k = A[i][k];
      for (j = 0; j < q; ++j)
        M[i][j] += temp_i_k * B[k][j];
    }
  }
```

VALGRIND:

```
==27704== I   refs:      41,014,766,985
==27704== I1  misses:             1,335
==27704== LLi misses:             1,324
==27704== I1  miss rate:           0.00%
==27704== LLi miss rate:           0.00%
==27704==
==27704== D   refs:      12,549,977,596  (10,646,160,736 rd   + 1,903,816,860 wr)
==27704== D1  misses:       109,537,935  (   109,355,373 rd   +       182,562 wr)
==27704== LLd misses:           457,616  (       275,111 rd   +       182,505 wr)
==27704== D1  miss rate:            0.9% (           1.0%     +           0.0%  )
==27704== LLd miss rate:            0.0% (           0.0%     +           0.0%  )
==27704==
==27704== LL refs:          109,539,270  (   109,356,708 rd   +       182,562 wr)
==27704== LL misses:            458,940  (       276,435 rd   +       182,505 wr)
==27704== LL miss rate:             0.0% (           0.0%     +           0.0%  )
```

PERF:

```
+   99.01%     0.00%  caching_in_loop  libc-2.17.so      [.] __libc_start_main
+   99.01%     0.00%  caching_in_loop  caching_in_loop   [.] main
+   95.88%    95.67%  caching_in_loop  caching_in_loop   [.] matmul
+    2.27%     0.08%  caching_in_loop  libc-2.17.so      [.] fprintf
+    1.23%     1.23%  caching_in_loop  libc-2.17.so      [.] vfprintf
     0.74%     0.74%  caching_in_loop  [kernel]          [k] 0xffffffffb8f75ed0
+    0.74%     0.00%  caching_in_loop  libc-2.17.so      [.] __GI___libc_write
+    0.62%     0.62%  caching_in_loop  libc-2.17.so      [.] _itoa_word
     0.33%     0.28%  caching_in_loop  caching_in_loop   [.] takeinput
     0.28%     0.28%  caching_in_loop  [kernel]          [k] 0xffffffffb8f6c4d6
     0.21%     0.21%  caching_in_loop  libc-2.17.so      [.] _IO_file_xsputn@@GLIBC_2.2.5
     0.20%     0.20%  caching_in_loop  libc-2.17.so      [.] __random_r
```

5. **Using pointers to access the elements of the array**, instead of directly accessing the value. Some compilers work faster when directly accessing the value by the use of pointers.
   Execution time for matmul = 2.430550
   Total Execution time = 2.704391

   Change:

   ```
   *(A + i); // better
    A[i];     // Slower
   ```

   CODE:

```
register int *pa = NULL;
register int *pb = NULL;
register int *pm = NULL;
for (i = 0; i < m; ++i){
  pa = *(A + i);
  pm = *(M + i);
  for (k = 0; k < n; ++k){
    temp_i_k = *(pa + k);
    pb = *(B + k);
    for (j = 0; j < q; ++j){
      *(pm + j) += temp_i_k * *(pb + j);
    }
  }
}
```

VALGRIND:

```
==27725== I   refs:      32,379,097,497
==27725== I1  misses:             1,326
==27725== LLi misses:             1,315
==27725== I1  miss rate:           0.00%
==27725== LLi miss rate:           0.00%
==27725==
==27725== D   refs:       7,365,979,879  (5,462,163,069 rd   + 1,903,816,810 wr)
==27725== D1  misses:       109,537,931  (  109,355,380 rd   +       182,551 wr)
==27725== LLd misses:           457,614  (      275,114 rd   +       182,500 wr)
==27725== D1  miss rate:            1.5% (          2.0%     +           0.0%  )
==27725== LLd miss rate:            0.0% (          0.0%     +           0.0%  )
==27725==
==27725== LL refs:          109,539,257  (  109,356,706 rd   +       182,551 wr)
==27725== LL misses:            458,929  (      276,429 rd   +       182,500 wr)
==27725== LL miss rate:             0.0% (          0.0%     +           0.0%  )
```

PERF:

```
+   99.45%     0.00%  pointers  libc-2.17.so      [.] __libc_start_main
+   99.45%     0.00%  pointers  pointers          [.] main
+   95.11%    94.98%  pointers  pointers          [.] matmul
+    3.27%     0.10%  pointers  libc-2.17.so      [.] fprintf
+    1.75%     1.75%  pointers  libc-2.17.so      [.] vfprintf
+    0.95%     0.95%  pointers  libc-2.17.so      [.] _itoa_word
     0.40%     0.34%  pointers  pointers          [.] takeinput
     0.39%     0.39%  pointers  libc-2.17.so      [.] _IO_file_xsputn@@GLIBC_2.2.5
     0.26%     0.26%  pointers  libc-2.17.so      [.] __strchrnul
     0.21%     0.21%  pointers  [kernel]          [k] 0xffffffffb8f6c4d6
     0.20%     0.20%  pointers  libc-2.17.so      [.] __random
     0.18%     0.18%  pointers  libc-2.17.so      [.] __random_r
     0.15%     0.15%  pointers  [kernel]          [k] 0xffffffffb8f75ed0
     0.15%     0.14%  pointers  pointers          [.] print_matrix
```

6. **Loop unrolling** - In this i try to remove or reduce iterations.It increases the program's speed by eliminating loop control instruction and loop test instructions. Instead of the normal innermost loop, I unrolled it into 16 lines executing in the same loop. So the no. of iterations of the innermost loop is decreased by a factor of 16.
   Execution time for matmul = 1.673551
   Total Execution time = 1.928794

   CODE:

```
register int *pb = NULL, *pr = M[0];
for (i = 0; i < m; ++i){
  for (k = 0; k < n; ++k){
    pb = B[k];
    temp_i_k = A[i][k];
    pr = M[i];
    for (j = 0; j < q - 15; j += 16){
      // M[i][j] += temp_i_k * B[k][j];
      *pr++ += temp_i_k * (*pb++);
      *pr++ += temp_i_k * (*pb++);
      *pr++ += temp_i_k * (*pb++);
      *pr++ += temp_i_k * (*pb++);
      *pr++ += temp_i_k * (*pb++);
```

```
        *pr++ += temp_i_k * (*pb++);
        *pr++ += temp_i_k * (*pb++);
        *pr++ += temp_i_k * (*pb++);
        *pr++ += temp_i_k * (*pb++);
        *pr++ += temp_i_k * (*pb++);
        *pr++ += temp_i_k * (*pb++);
        *pr++ += temp_i_k * (*pb++);
        *pr++ += temp_i_k * (*pb++);
        *pr++ += temp_i_k * (*pb++);
        *pr++ += temp_i_k * (*pb++);
        *pr++ += temp_i_k * (*pb++);
      }
      for (; j < q; ++j){
        *pr++ += temp_i_k * (*pb++);
      }
    }
  }
```

VALGRIND:

```
==27734== I   refs:      19,115,247,825
==27734== I1  misses:             1,333
==27734== LLi misses:             1,322
==27734== I1  miss rate:           0.00%
==27734== LLi miss rate:           0.00%
==27734==
==27734== D   refs:       5,750,297,452  (3,846,480,643 rd   + 1,903,816,809 wr)
==27734== D1  misses:       109,537,935  (  109,355,373 rd   +       182,562 wr)
==27734== LLd misses:           457,616  (      275,111 rd   +       182,505 wr)
==27734== D1  miss rate:            1.9% (          2.8%     +           0.0%  )
==27734== LLd miss rate:            0.0% (          0.0%     +           0.0%  )
==27734==
==27734== LL refs:          109,539,268  (  109,356,706 rd   +       182,562 wr)
==27734== LL misses:            458,938  (      276,433 rd   +       182,505 wr)
==27734== LL miss rate:             0.0% (          0.0%     +           0.0%  )
```

PERF:

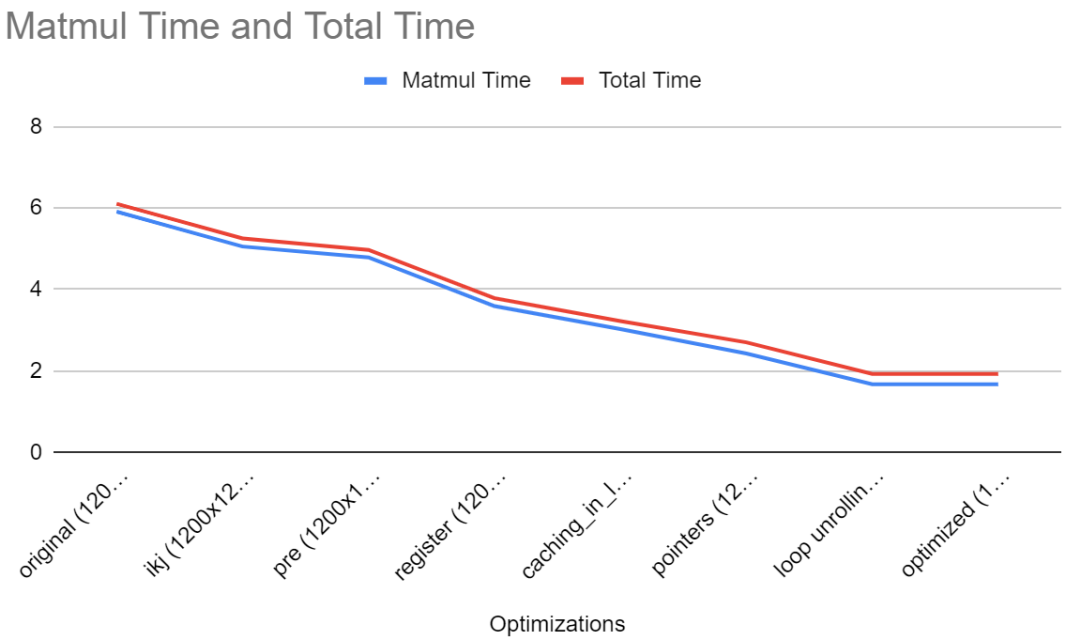
```
+   98.59%     0.00%  loop_unrolling  libc-2.17.so      [.] __libc_start_main
+   98.59%     0.00%  loop_unrolling  loop_unrolling    [.] main
+   94.27%    93.78%  loop_unrolling  loop_unrolling    [.] matmul
+    3.12%     0.06%  loop_unrolling  libc-2.17.so      [.] fprintf
+    1.73%     1.73%  loop_unrolling  libc-2.17.so      [.] vfprintf
     0.96%     0.96%  loop_unrolling  [kernel]          [k] 0xffffffffb8f75ed0
+    0.95%     0.00%  loop_unrolling  libc-2.17.so      [.] __GI___libc_write
+    0.90%     0.90%  loop_unrolling  libc-2.17.so      [.] _itoa_word
+    0.58%     0.58%  loop_unrolling  [kernel]          [k] 0xffffffffb8f6c4d6
     0.48%     0.42%  loop_unrolling  loop_unrolling    [.] takeinput
     0.34%     0.33%  loop_unrolling  libc-2.17.so      [.] _IO_file_xsputn@@GLIBC_2.2.5
     0.25%     0.25%  loop_unrolling  libc-2.17.so      [.] __random_r
     0.24%     0.24%  loop_unrolling  libc-2.17.so      [.] __strchrnul
```

Summary:

The code is optimized sequentially by taking the above steps in order.
Here is a table with the relevant stats:

	Matmul Time	Total Time	Perf % for matmul	Number of L1 Cache Miss*	Cache Miss Rate*
original (3500x3500)	310.477654	312.193961	98.11	++	8.7
original (1200x1200)	5.914727	6.106516	98.11	1.83E+10	8.7
ikj (1200x1200)	5.057156	5.258106	97.41	1.10E+09	0.3
pre (1200x1200)	4.790539	4.973955	97.6	1.10E+09	0.3
register (1200x1200)	3.593632	3.787293	96.7	1.10E+09	0.7
caching_in_loop (1200x1200)	3.029244	3.223919	95.88	1.10E+09	0.9
pointers (1200x1200)	2.43055	2.704391	95.11	1.10E+09	1.5
loop unrolling (1200x1200)	1.673551	1.928794	94.27	1.10E+09	1.9
optimized (1200x1200)	1.673551	1.928794	94.27	1.10E+09	1.9
optimized (3500x3500)	31.956461	33.564124	94.27	2.7E+09	2

++ : the process takes a huge amount of time to simulate on Valgrind, so skipped this.
NOTE*: The cache miss rate seems to be increasing, that is because in the optimized codes, the total references made to the cache is even lower. The point to note is the Number of Cache Miss, is the same from after the second level of optimization.



Task 2: Merge Sort

Naive approach: Recursive Merge Sort

- Runtime for the original code with N = 6e10, (N = size of the array)
  Execution time for MergeSort = 1.301503
  Total Execution time = 1.937340

- VALGRIND:

```
==29243== I   refs:      11,681,092,958
==29243== I1  misses:             1,331
==29243== LLi misses:             1,320
==29243== I1  miss rate:           0.00%
==29243== LLi miss rate:           0.00%
==29243==
==29243== D   refs:       5,295,286,564  (4,208,315,600 rd   + 1,086,970,964 wr)
==29243== D1  misses:        17,566,733  (    8,888,295 rd   +     8,678,438 wr)
==29243== LLd misses:         5,569,547  (    2,902,285 rd   +     2,667,262 wr)
==29243== D1  miss rate:            0.3% (          0.2%     +           0.8%   )
==29243== LLd miss rate:            0.1% (          0.1%     +           0.2%   )
==29243==
==29243== LL refs:           17,568,064  (    8,889,626 rd   +     8,678,438 wr)
==29243== LL misses:          5,570,867  (    2,903,605 rd   +     2,667,262 wr)
==29243== LL miss rate:             0.0% (          0.0%     +           0.2%   )
```

- PERF:

```
+   95.63%     0.00%  original  libc-2.17.so      [.] __libc_start_main
+   95.62%     0.00%  original  original          [.] main
+   69.09%     3.54%  original  original          [.] mergeSort
+   65.55%    65.37%  original  original          [.] merge
+   21.63%     0.48%  original  libc-2.17.so      [.] fprintf
+   12.24%    12.21%  original  libc-2.17.so      [.] vfprintf
+    5.31%     5.30%  original  libc-2.17.so      [.] _itoa_word
+    2.74%     2.74%  original  libc-2.17.so      [.] _IO_file_xsputn@@GLIBC_2.2.5
+    1.85%     1.85%  original  libc-2.17.so      [.] __strchrnul
+    1.73%     1.73%  original  original          [.] printArrayFile
     1.42%     1.42%  original  [kernel]          [k] 0xffffffffb9d75ed0
+    1.41%     0.00%  original  libc-2.17.so      [.] __GI___libc_write
+    1.35%     1.31%  original  original          [.] takeInput
```

Optimizations :

The following optimizations were tried out in the order mentioned below.

1. Using **iterative Merge Sort** implementation instead of the original recursive one. This saves time as it avoids the excessive recursive function calls and thus the overheads are avoided.
   Execution time for Iterative MergeSort = 1.275348
   Total Execution time = 1.969982

   CODE:

```
void iter_mergeSort(int arr[], int n){
    int curr_size;
    int l;
    for (curr_size = 1; curr_size <= n - 1; curr_size *= 2){
```

```
        for (l = 0; l < n - 1; l += 2 * curr_size){
            int m = min(l + curr_size - 1, n - 1);
            int r = min(l + 2 * curr_size - 1, n - 1);
            merge(arr, l, m, r);
        }
    }
}
```

Perf comparison showing difference between recursive and iterative.

```
+   97.41%     0.00%  rec_iter  libc-2.17.so      [.] __libc_start_main
+   97.41%     0.53%  rec_iter  rec_iter          [.] main
+   77.04%    76.89%  rec_iter  rec_iter          [.] merge
+   40.64%     2.22%  rec_iter  rec_iter          [.] rec_mergeSort
+   40.29%     1.01%  rec_iter  rec_iter          [.] iter_mergeSort
+   12.93%     0.24%  rec_iter  libc-2.17.so      [.] fprintf
```

We can see that iterative was faster than the recursive one, but only by a fraction, as the most time is taken up by the *Merge* operation which is common in both of them. Although it's faster in speed due to absence of calls to function stack, in terms of Cache optimization, Iterative is worse than Recursive approach.

VALGRIND:

```
==30051== I   refs:      11,764,584,296
==30051== I1  misses:             1,335
==30051== LLi misses:             1,324
==30051== I1  miss rate:           0.00%
==30051== LLi miss rate:           0.00%
==30051==
==30051== D   refs:       5,353,903,014  (4,268,154,581 rd   + 1,085,748,433 wr)
==30051== D1  misses:        21,867,979  (   13,126,555 rd   +     8,741,424 wr)
==30051== LLd misses:        12,872,943  (   10,105,782 rd   +     2,767,161 wr)
==30051== D1  miss rate:            0.4% (          0.3%     +           0.8%   )
==30051== LLd miss rate:            0.2% (          0.2%     +           0.3%   )
==30051==
==30051== LL refs:           21,869,314  (   13,127,890 rd   +     8,741,424 wr)
==30051== LL misses:         12,874,267  (   10,107,106 rd   +     2,767,161 wr)
==30051== LL miss rate:             0.1% (          0.1%     +           0.3%   )
```

PERF:

```
+   95.44%     0.00%  iterative  libc-2.17.so      [.] __libc_start_main
+   95.44%     0.00%  iterative  iterative         [.] main
+   68.66%     1.78%  iterative  iterative         [.] iter_mergeSort
+   65.71%    65.54%  iterative  iterative         [.] merge
+   21.66%     0.49%  iterative  libc-2.17.so      [.] fprintf
+   12.32%    12.32%  iterative  libc-2.17.so      [.] vfprintf
+    5.58%     5.58%  iterative  libc-2.17.so      [.] _itoa_word
+    2.70%     2.70%  iterative  libc-2.17.so      [.] _IO_file_xsputn@@GLIBC_2.2.5
+    1.64%     1.62%  iterative  iterative         [.] printArrayFile
     1.62%     1.62%  iterative  [kernel]          [k] 0xffffffffb9d75ed0
+    1.61%     0.00%  iterative  libc-2.17.so      [.] __GI___libc_write
+    1.59%     1.59%  iterative  libc-2.17.so      [.] __strchrnul
+    1.43%     1.43%  iterative  iterative         [.] min
```

2. **“Register” keyword with the variables**. This makes the variable to be stored in the register directly. This is helpful when the variable is used very frequently, for example as a counter in a loop.
Execution time for Iterative MergeSort = 1.008754
Total Execution time = 1.621189

   Change:

```
register int i = 0;      // Preferred
int i = 0;               // previous
```

```
// Example
register int i = 0;
register int j = 0;
```

   VALGRIND:

```
==30109== I   refs:      10,512,490,877
==30109== I1  misses:             1,336
==30109== LLi misses:             1,325
==30109== I1  miss rate:           0.00%
==30109== LLi miss rate:           0.00%
==30109==
==30109== D   refs:       3,166,445,505  (2,104,697,116 rd   + 1,061,748,389 wr)
==30109== D1  misses:        21,856,271  (   13,122,165 rd   +     8,734,106 wr)
==30109== LLd misses:        12,872,948  (   10,105,784 rd   +     2,767,164 wr)
==30109== D1  miss rate:            0.7% (          0.6%     +           0.8%  )
==30109== LLd miss rate:            0.4% (          0.5%     +           0.3%  )
==30109==
==30109== LL refs:           21,857,607  (   13,123,501 rd   +     8,734,106 wr)
==30109== LL misses:         12,874,273  (   10,107,109 rd   +     2,767,164 wr)
==30109== LL miss rate:             0.1% (          0.1%     +           0.3%  )
```

   PERF:

```
+   94.09%     0.00%  register  libc-2.17.so      [.] __libc_start_main
+   94.08%     0.00%  register  register          [.] main
+   60.99%     1.20%  register  register          [.] iter_mergeSort
+   57.96%    57.67%  register  register          [.] merge
+   27.73%     0.56%  register  libc-2.17.so      [.] fprintf
+   16.13%    16.10%  register  libc-2.17.so      [.] vfprintf
+    6.83%     6.83%  register  libc-2.17.so      [.] _itoa_word
+    3.50%     3.48%  register  libc-2.17.so      [.] _IO_file_xsputn@@GLIBC_2.2.5
+    2.18%     2.18%  register  libc-2.17.so      [.] __strchrnul
     2.10%     2.10%  register  [kernel]          [k] 0xffffffffb8f75ed0
+    2.10%     0.00%  register  libc-2.17.so      [.] __GI___libc_write
+    1.98%     1.98%  register  register          [.] min
+    1.61%     1.52%  register  register          [.] takeInput
+    1.49%     1.49%  register  libc-2.17.so      [.] __GI___mempcpy
```

3. **Converting all post-increment/decrement to pre-increment/decrement operator.**

```
++i; // Preferred
i++; // Slower
```

   This is faster because in case of post op, the value is first copied and then the operation takes place, whereas in case of pre-op, there is no overhead of copying the variable value as the variable is instantly updated without the need of previous value.
Execution time for Iterative MergeSort = 0.962132
Total Execution time = 1.600874

VALGRIND: (same as previous)

PERF:

```
+   93.98%     0.00%  pre      libc-2.17.so      [.] __libc_start_main
+   93.98%     0.00%  pre      pre               [.] main
+   60.88%     1.14%  pre      pre               [.] iter_mergeSort
+   57.69%    57.46%  pre      pre               [.] merge
+   27.45%     0.53%  pre      libc-2.17.so      [.] fprintf
+   15.86%    15.83%  pre      libc-2.17.so      [.] vfprintf
+    6.70%     6.70%  pre      libc-2.17.so      [.] _itoa_word
+    3.24%     3.23%  pre      libc-2.17.so      [.] _IO_file_xsputn@@GLIBC_2.2.5
+    2.39%     2.38%  pre      libc-2.17.so      [.] __strchrnul
     2.26%     2.26%  pre      [kernel]          [k] 0xffffffffb8f75ed0
+    2.26%     2.25%  pre      pre               [.] min
+    2.25%     0.01%  pre      libc-2.17.so      [.] __GI___libc_write
+    1.60%     1.52%  pre      pre               [.] takeInput
+    1.55%     1.54%  pre      libc-2.17.so      [.] __GI___mempcpy
+    1.40%     1.40%  pre      pre               [.] printArrayFile
```

4. **Using pointers to access the elements of the array**, instead of directly accessing the value. Some compilers work faster when directly accessing the value by the use of pointers.
   And also using *memcpy()* in order to copy the subarrays and sort them individually, the memcpy function has been used to optimize the run time.
   Execution time for Iterative MergeSort = 0.857645
   Total Execution time = 1.503787
   Change:

```
*(A + i); // better
 A[i];    // Slower
```

   CODE:

```
// for (i = 0; i < n1; ++i) L[i] = arr[l + i];
memcpy(L, &arr[l], sizeof(int) * n1);
// for (j = 0; j < n2; ++j) R[j] = arr[m + 1 + j];
memcpy(R, &arr[m + 1], sizeof(int) * n2);
```

   VALGRIND: (same as previous)

   PERF:

```
+   93.92%     0.00%  pointers  libc-2.17.so      [.] __libc_start_main
+   93.92%     0.00%  pointers  pointers          [.] main
+   60.97%     1.28%  pointers  pointers          [.] iter_mergeSort
+   51.38%    51.29%  pointers  pointers          [.] merge
+   27.58%     0.61%  pointers  libc-2.17.so      [.] fprintf
+   15.83%    15.79%  pointers  libc-2.17.so      [.] vfprintf
+    6.80%     6.79%  pointers  libc-2.17.so      [.] _itoa_word
+    6.77%     6.62%  pointers  libc-2.17.so      [.] __memcpy_ssse3_back
+    3.60%     3.59%  pointers  libc-2.17.so      [.] _IO_file_xsputn@@GLIBC_2.2.5
+    2.28%     0.01%  pointers  libc-2.17.so      [.] __GI___libc_write
     2.28%     2.28%  pointers  [kernel]          [k] 0xffffffffb8f75ed0
+    2.08%     2.07%  pointers  libc-2.17.so      [.] __strchrnul
+    1.59%     1.58%  pointers  pointers          [.] min
+    1.55%     1.47%  pointers  pointers          [.] takeInput
+    1.45%     1.44%  pointers  libc-2.17.so      [.] __GI___mempcpy
+    1.24%     1.22%  pointers  pointers          [.] printArrayFile
```

5. **Trying Insertion Sort for small values of N (<=32)**.
   Tried using insertion sort for small values of N, to increase performance, and Merge sort will only start working for size greater than this value. This saves time as in merge sort, time is wasted in accessing extra arrays and excess comparisons.
   Execution time for Iterative MergeSort = 0.749808
   Total Execution time = 1.409480

   CODE:

```
    register int iter1, temp, j;
    int insertion_lim = 30;
    for (iter1 = 0; iter1 < n; iter1 += insertion_lim){
        // register int left = iter1;
        register int right = min(iter1 + insertion_lim - 1, n - 1);
        register int iter2;
        for (iter2 = iter1 + 1; iter2 <= right; ++iter2){
            temp = *(arr + iter2);
            j = iter2 - 1;
            while (*(arr + j) > temp && j >= iter1){
                *(arr + j + 1) = *(arr + j);
                --j;
            }
            *(arr + j + 1) = temp;
        }
    }
```

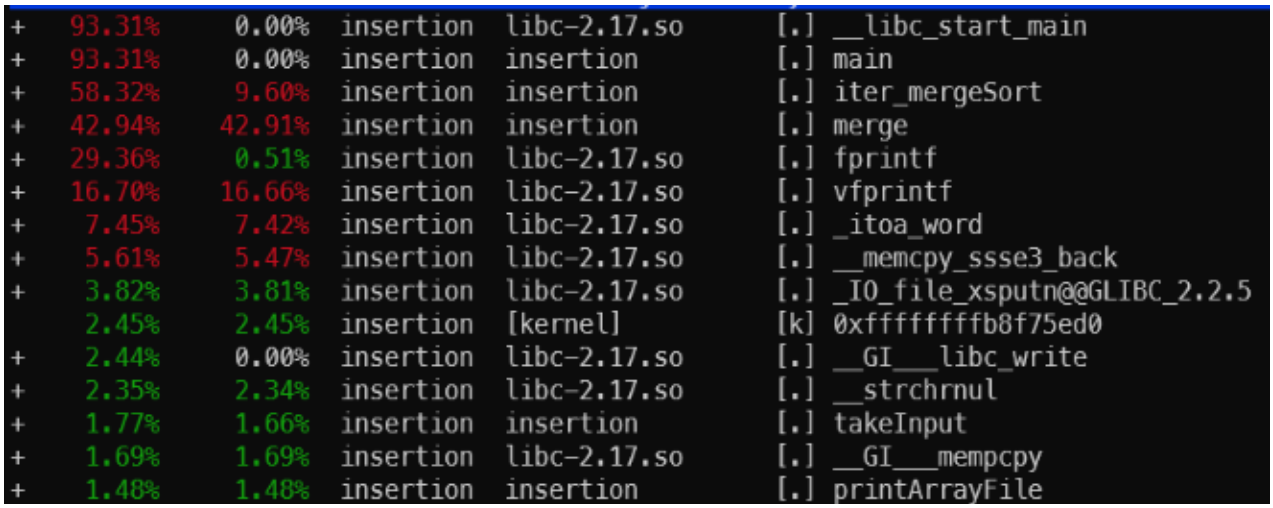
Below is a table showing time results by using different limits. As we can see the N<=30 case worked best.

Value of limit	**20**	**30**	**40**	**50**	**60**
Average time	0.765664	0.7547165	0.7917615	0.793422	0.825713

VALGRIND: (improved slightly than before)

```
==30702== I   refs:      8,946,562,133
==30702== I1  misses:            1,348
==30702== LLi misses:            1,337
==30702== I1  miss rate:          0.00%
==30702== LLi miss rate:          0.00%
==30702==
==30702== D   refs:      3,015,062,018  (1,934,310,191 rd   + 1,080,751,827 wr)
==30702== D1  misses:       20,303,679  (   11,598,415 rd   +     8,705,264 wr)
==30702== LLd misses:       11,277,898  (    8,568,454 rd   +     2,709,444 wr)
==30702== D1  miss rate:           0.7% (          0.6%     +           0.8%  )
==30702== LLd miss rate:           0.4% (          0.4%     +           0.3%  )
==30702==
==30702== LL refs:          20,305,027  (   11,599,763 rd   +     8,705,264 wr)
==30702== LL misses:        11,279,235  (    8,569,791 rd   +     2,709,444 wr)
==30702== LL miss rate:            0.1% (          0.1%     +           0.3%  )
```

PERF:

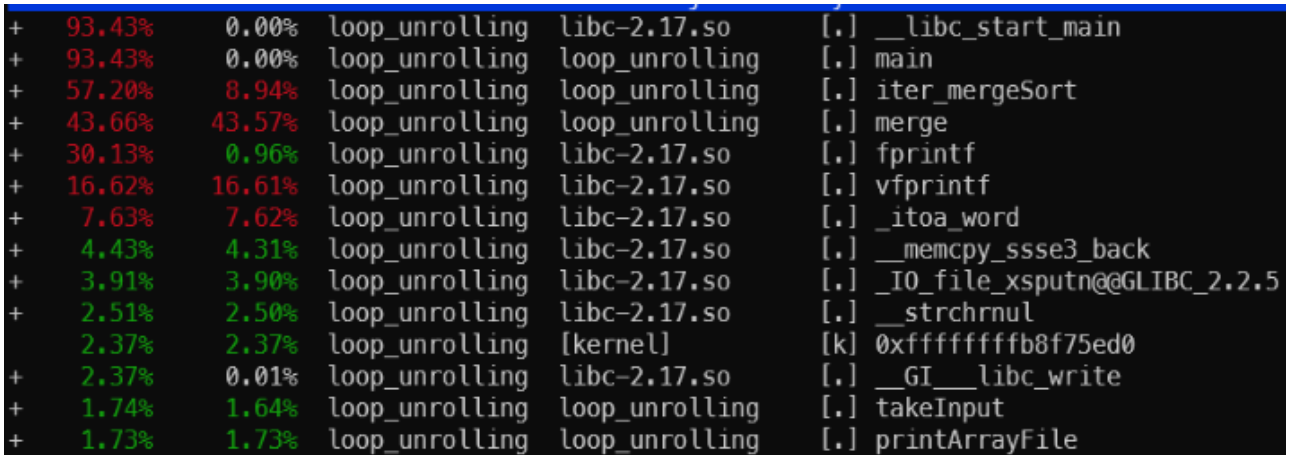


+	93.31%	0.00%	insertion	libc-2.17.so	[.] __libc_start_main
+	93.31%	0.00%	insertion	insertion	[.] main
+	58.32%	9.60%	insertion	insertion	[.] iter_mergeSort
+	42.94%	42.91%	insertion	insertion	[.] merge
+	29.36%	0.51%	insertion	libc-2.17.so	[.] fprintf
+	16.70%	16.66%	insertion	libc-2.17.so	[.] vfprintf
+	7.45%	7.42%	insertion	libc-2.17.so	[.] _itoa_word
+	5.61%	5.47%	insertion	libc-2.17.so	[.] __memcpy_ssse3_back
+	3.82%	3.81%	insertion	libc-2.17.so	[.] _IO_file_xsputn@@GLIBC_2.2.5
	2.45%	2.45%	insertion	[kernel]	[k] 0xffffffffb8f75ed0
+	2.44%	0.00%	insertion	libc-2.17.so	[.] __GI___libc_write
+	2.35%	2.34%	insertion	libc-2.17.so	[.] __strchrnul
+	1.77%	1.66%	insertion	insertion	[.] takeInput
+	1.69%	1.69%	insertion	libc-2.17.so	[.] __GI___mempcpy
+	1.48%	1.48%	insertion	insertion	[.] printArrayFile

6. **Loop unrolling** - In this i try to remove or reduce iterations.It increases the program's speed by eliminating loop control instruction and loop test instructions. Instead of the normal innermost loop, I unrolled it into 16 lines executing in the same loop. So the no. of iterations of the innermost loop is decreased by a factor of 16. Loop unrolling was only slightly better than the previous one, and in some cases even worse.
   Execution time for Iterative MergeSort = 0.738545
   Total Execution time = 1.408642

   VALGRIND: (similar to previous one, with very little improvement)

   PERF:



+	93.43%	0.00%	loop_unrolling	libc-2.17.so	[.] __libc_start_main
+	93.43%	0.00%	loop_unrolling	loop_unrolling	[.] main
+	57.20%	8.94%	loop_unrolling	loop_unrolling	[.] iter_mergeSort
+	43.66%	43.57%	loop_unrolling	loop_unrolling	[.] merge
+	30.13%	0.96%	loop_unrolling	libc-2.17.so	[.] fprintf
+	16.62%	16.61%	loop_unrolling	libc-2.17.so	[.] vfprintf
+	7.63%	7.62%	loop_unrolling	libc-2.17.so	[.] _itoa_word
+	4.43%	4.31%	loop_unrolling	libc-2.17.so	[.] __memcpy_ssse3_back
+	3.91%	3.90%	loop_unrolling	libc-2.17.so	[.] _IO_file_xsputn@@GLIBC_2.2.5
+	2.51%	2.50%	loop_unrolling	libc-2.17.so	[.] __strchrnul
	2.37%	2.37%	loop_unrolling	[kernel]	[k] 0xffffffffb8f75ed0
+	2.37%	0.01%	loop_unrolling	libc-2.17.so	[.] __GI___libc_write
+	1.74%	1.64%	loop_unrolling	loop_unrolling	[.] takeInput
+	1.73%	1.73%	loop_unrolling	loop_unrolling	[.] printArrayFile

8. **Register in functions:** Using the register keyword for the function arguments as well, specially when passing the array as the argument. Earlier I used the register keyword with only loop counters.
Execution time for Iterative MergeSort = 0.694686
Total Execution time = 1.353260

Change:

```
void printArray(int A[], int size)                    // before
void printArray(register int *A, register int size)   // now
```

VALGRIND: (very little improvement than before)

```
==8911== Command: ./register_with_array
==8911==
--8911-- warning: L3 cache found, using its data for the LL simulation.

Execution time for Iterative MergeSort = 11.250492
Total Execution time  = 28.494523
==8911==
==8911== I   refs:      8,691,262,236
==8911== I1  misses:            1,370
==8911== LLi misses:            1,365
==8911== I1  miss rate:          0.00%
==8911== LLi miss rate:          0.00%
==8911==
==8911== D   refs:      2,848,126,162  (1,767,374,332 rd   + 1,080,751,830 wr)
==8911== D1  misses:       20,298,584  (   11,595,670 rd   +     8,702,914 wr)
==8911== LLd misses:       11,277,884  (    8,568,445 rd   +     2,709,439 wr)
==8911== D1  miss rate:           0.7% (          0.7%     +           0.8%   )
==8911== LLd miss rate:           0.4% (          0.5%     +           0.3%   )
==8911==
==8911== LL refs:          20,299,954  (   11,597,040 rd   +     8,702,914 wr)
==8911== LL misses:        11,279,249  (    8,569,810 rd   +     2,709,439 wr)
==8911== LL miss rate:            0.1% (          0.1%     +           0.3%   )
```

PERF:

```
+   93.18%     0.00%  register_with_a  libc-2.17.so          [.] __libc_start_main
+   93.18%     0.00%  register_with_a  register_with_array   [.] main
+   57.62%     8.97%  register_with_a  register_with_array   [.] iter_mergeSort
+   43.75%    43.74%  register_with_a  register_with_array   [.] merge
+   29.29%     1.55%  register_with_a  libc-2.17.so          [.] fprintf
+   16.23%    16.22%  register_with_a  libc-2.17.so          [.] vfprintf
+    6.69%     6.69%  register_with_a  libc-2.17.so          [.] _itoa_word
+    4.82%     4.02%  register_with_a  libc-2.17.so          [.] __memcpy_ssse3_back
+    3.71%     3.71%  register_with_a  libc-2.17.so          [.] _IO_file_xsputn@@GLIBC_2.2.5
+    2.74%     2.74%  register_with_a  libc-2.17.so          [.] __strchrnul
     2.70%     2.70%  register_with_a  [unknown]             [k] 0xffffffff99775ee0
+    2.68%     0.00%  register_with_a  libc-2.17.so          [.] __GI___libc_write
+    2.35%     1.69%  register_with_a  register_with_array   [.] takeInput
+    1.50%     1.50%  register_with_a  [unknown]             [k] 0xffffffff9976c4e7
+    1.50%     1.50%  register_with_a  libc-2.17.so          [.] __GI___mempcpy
+    1.18%     1.18%  register_with_a  register_with_array   [.] printArrayFile
```

Summary:

The code is optimized sequentially by taking the above steps in order.
Here is a table with the relevant stats

Optimizations	Merge Sort Time	Total Time	Perf % for MergeSort
original	1.301503	1.93734	69.09
iterative	1.275348	1.969982	68.66
register	1.008754	1.621189	60.99
pre	0.962132	1.600874	60.88
pointers	0.857645	1.503787	60.97
insertion sort	0.749808	1.40948	58.32
loop unrolling	0.738545	1.408642	57.2
register in functions	0.694686	1.35326	57.6
optimized	0.694686	1.35326	57.6

Points to note:

- The cache miss and references are increased as compared to the original (recursive) mergesort. But the overall speed of the new optimized code is much faster. So the code is optimized more towards speed.
- In the optimized code, the Perf % is higher than the previous one, that is because some other functions are optimized more than before (example fprintf, which is a function used to print array to file). So the whole code is faster, but the Merge Sort function is the same as the previous one.